LEGIBILITY NOTICE

A major purpose of the Technical Information Center is to provide the broadest dissemination possible of information contained in DOE's Research and Development Reports to business, industry, the academic community, and federal, state and local governments.

Although a small portion of this report is not reproducible, it is being made available to expedite the availability of information on the research discussed herein.

LA-UR -86-3300

B/P

LA-UR--86-3300

DE88 009178

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36

TITLE BENCHMARKING THE IBM 3090 WITH VECTOR FACILITY

AUTHOR(S) Ralph G. Brickner
Harvey J. Wasserman
Ann H. Hayes
James W. Moore

SUBMITTED TO For distribution upon request

DISCLAIMER

Los Alamos National Laboratory
Los Alamos, New Mexico 87545

BENCHMARKING THE IBM 3090 WITH VECTOR FACILITY

Ralph G. Brickner
Harvey J. Wasserman
Ann H. Hayes
James W. Moore

Computing and Communications Division
Los Alamos National Laboratory

I. INTRODUCTION

The IBM 3090 with Vector Facility is IBM's first foray into the scientific vector processor market since Cray Research Inc. created that market in the mid 1970s. The appearance of the 3090 both legitimizes vector processing to the masses and serves notice to Cray that IBM intends to be a real presence in the high-end scientific computing arena. It is, therefore, natural that we at Los Alamos are very interested in the performance of this machine. Over the past few months, we have worked with IBM to obtain an accurate benchmark of the machine. The 3090 is a complex, shared-memory, multiprocessor system, and we have attempted to address at least the major components of the system performance in this benchmark: scalar speed, vector speed, compiler performance, and the Multitasking Facility. As usual, the benchmarks are intended to measure computational performance on a dedicated system. In particular, this means no I/O or throughput measurements were made.

We urge the reader to keep the following points in mind while reading this report. First, the 3090 is a general-purpose, commercial mainframe. It is not intended to be a state-of-the-art scientific supercomputer. However, it *is* intended to be a high performance computer, so IBM has concentrated on making its scalar performance very good. This has been accomplished by evolutionary, not revolutionary, development from previous IBM mainframes, the 308X series. The second point to keep in mind is that the Vector Facility is an add-on piece of hardware. The 3090 was designed for the commercial marketplace where reasonable speed is to be obtained at reasonable cost. The design of the Vector Facility was driven by the need to fit in with existing hardware, not by the desire to build a superfast vector machine. In fact, the Vector Facility increases the 3090 cost by only 10-15%, while performance increases may be much greater.

In this report, we will first describe the hardware and software currently comprising the 3090 system. Next, we will discuss the objectives and methodology of our benchmarking trip to Gaithersburg, Maryland, and Kingston, New York. We then present the actual benchmark results, along with the comparable numbers for the CRAY X-MP. A discussion of the results follows, along with an analysis of hardware and compiler features and performance. A brief section describes the results of the Multitasking Facility benchmark and the multitasking primitives. Finally, we present our conclusions on the suitability of this machine for our workload.

II. IBM 3090 SYSTEM ARCHITECTURE

Currently, the IBM 3090 is available in four models: the single-processor Models 150 and 180, the dyadic Model 200, and the four-processor Model 400. Since the single-processor models lack multitasking capability, we will describe only the multiprocessor models.

A. The 3090 Central Processor

The 3090 central processor has an 18.5 ns cycle time and is constructed with emitter coupled logic technology and IBM's thermal conduction module packaging. The processor is microcode controlled and consists of several elements. The instruction element decodes instructions, calculates addresses, sends fetch requests to storage control, and provides the execution element with operation codes, operands, and operand addresses. The execution element processes instructions and interrupts, initiates control functions, and performs the scalar logical and arithmetic computations. The instruction and execution elements are pipelined and operate in parallel. A third element, the control storage element, provides the instruction and execution elements with microcode and contains control storage and the processor registers. Finally, a buffer control element handles all main memory requests and contains the 64-kbyte high-speed cache, cache management hardware, and virtual memory support hardware.

Several features of the 3090 processor contribute to high speed scalar arithmetic performance. First, a separate fixed- and floating-point multiply unit accelerates these computations. Second, a new technique was employed for floating point add/subtract, saving one machine cycle per calculation over previous IBM designs. Third, the processor has special circuitry to reduce overhead in inner DO loops. Finally, all relevant internal data paths, as well as all data paths in the storage hierarchy, are multiples of 64 bits.

B. Memory Hierarchy

The 3090 has a three-level storage hierarchy. The fastest storage is the 64-kbyte "write-in" cache contained in each central processor. This cache is addressable only by the processor in which it lies, but cache-to-cache transfers may be made between processors to improve performance. Transfers between cache and central storage are in 128-byte blocks. The second level is the shared central storage. Central storage employs a SECDED scheme, and defective memory may be deallocated in 4-kbyte increments. A 7-bit storage protection key provides hardware storage protection in 4-kbyte increments. The final level in the hierarchy is an optional expanded storage. The expanded storage is intended to reduce the paging and swapping I/O overhead incurred by the virtual memory and multiprocessing. Transfers between central and expanded storage are done in 4 kbyte pages. Expanded storage can detect and correct single- and double-bit errors and can detect triple bit and some multiple-bit errors. Central storage is a standard 64 Mbyte on the Model 200 and 128 Mbyte on the Model 400. Expanded storage of 64 Mbyte or 128 Mbyte may be installed on the Model 200, while 128 Mbyte or 256 Mbyte may be installed on the Model 400.

C. The 3090 Vector Facility

The Vector Facility is a field upgradable addition to the 3090 central processor. Any or all of the processors in a 3090 system may have attached Vector Facilities, and each Vector Facility serves as an extension of the processors' instruction and execution elements. Data types supported in hardware are binary (fixed-point) and 32- and 64-bit floating point numbers. There are sixteen 32-bit vector registers, each containing 128 elements, which may be configured as eight 64-bit registers. These registers read from and write to the cache. There are two vector functional units:

add logical and multiply divide. Compound multiply and add, multiply and subtract, and multiply and accumulate instructions drive both functional units generating two floating-point results per clock for an absolute peak rate of 108 MFLOPS* per processor. A total of 171 new instructions are provided for driving the Vector Facility. These instructions include vector add, subtract, multiply, divide, and compare. Operands may be in storage or in vector or scalar registers, and results go to vector or scalar registers. Addressing modes for vectors in storage include constant stride, indirect addressing (vector of indices), and masked selection. In addition to these and the compound instructions, there are vector count instructions and save, restore, and clear vector register instructions. A design goal of the Vector Facility was to provide a maximum of four times performance increase over the 3090 scalar speeds.

D. The I/O Subsystem

The 3090 Model 200 provides 32 (or optionally, 40 or 48) I/O channels; the Model 400 provides twice those numbers. Up to four channels (Model 200) or eight channels (Model 400) may be configured for byte multiplexing. All others operate in block-multiplex mode. Block-multiplex operation results in a maximum data transfer rate of 3 Mbyte/s. The channels are controlled by a channel control element (CCE), which is a separate microcode-controlled processor. The Model 200 has one CCE, the 400 has two CCEs. The CCE allows any processor to access any channel under system software control. Finally, a system control element arbitrates storage requests among the central processors and the I/O subsystem.

III. IBM 3090 SOFTWARE

Two operating systems are available for the 3090: MVS/SP (Multiple Virtual System/System Product) and VM/SP HPO (Virtual Machine System Product, High-Performance Option). Time-Sharing Option (TSO) is available for MVS to provide interactive capabilities. MVS is primarily batch oriented and system communication is by means of Job Control Language. VM supports a variety of guest operating systems including Conversational Monitor System (CMS), an interactive virtual operating system.

Software support for vectorization and multitasking is provided by VS Fortran, a superset of Fortran 77. VS Fortran provides three levels of scalar optimization and two levels of vectorization. Compiler output includes fairly detailed analysis of source code for vectorization. The compiler may choose to vectorize one of several nested loops based on its analysis. One feature of the vectorizing compiler is an economy analyzer, which attempts to evaluate the relative speed of scalar and vector versions of a compiled loop, generating code for the fastest one. In the future, IBM plans to offer compiler directives to aid in vectorization (although they are currently not available in a product). Multitasking has been implemented with three subroutine calls instead of Fortran language extensions.

Supporting software includes a hot spot analyzer (profiler), a source-level interactive debugger, and an extensive set of mathematical subroutines called the Engineering and Scientific Subroutine Library (ESSL). ESSL includes single- and double-precision routines and vector and scalar versions. Subroutines include linear equation solution, eigensystem analysis, signal processing applications, and random number generation.

*One MFLOPS equals one million floating point operations per second.

IV. THE BENCHMARKING TRIP

On April 15-16, 1986, members of the C-3 benchmark team performed tests on the IBM 3090/200 at the IBM Washington Systems Center in Gaithersburg, Maryland. The standard Los Alamos benchmark set was run on a two-processor system with 64 Mbytes of memory using the MVS operating system. A description of the programs in the set can be found in Reference 1. On April 18, further tests were run at IBM in Kingston, New York. The purpose of this second phase was to test IBM's Multitasking Facility (MTF) and also to investigate the effect of compiler directives for vectorization available only on a version of the compiler at Kingston. The machine at Kingston was also a 3090/200. At both sites, we ran on dedicated systems using Version 2.1.0 (January 1986) of the Fortran Compiler. IBM personnel later ran some of our codes using an older version of the compiler (Version 1.1.0 of September 1985); these results are discussed below. It was necessary to use the VS Fortran "AD(DBLPAD4)" compiler option to select 64-bit floating-point words on all of the codes.

In Gaithersburg, all programs were run in both scalar and vector mode to determine the effect of the Vector Facility on performance. Both scalar and vector numbers are reported. We also investigated an additional feature of the 3090 architecture at Gaithersburg: one of the processors in the Model 200 has a direct path from memory to its cache bypassing the system control element, which ordinarily handles memory requests. This bypass is known as the "fast path" to the processor's cache. Surmising that this processor would show improved performance in vector processing, we attempted to observe this effect by running the benchmarks first with both processors enabled and then with either one or the other processor enabled. We observed a slight degradation in performance (less than 5%) when using only one processor, attributing this to the fact that calls to the disabled processor had to be rerouted back to the enabled processor. Virtually no difference in timing was observed when either processor was used alone. We conclude that use of the direct memory-to-register path had no significant effect on the codes we ran.

V. BENCHMARKING RESULTS

A. Standard Benchmarks

Table I shows timings in both vector and scalar mode for the standard benchmark set. For comparison purposes, timings from a single processor of a CRAY X-MP/48 are given. Effectively, two levels of vectorization are available on the 3090: Level 1 performs vectorization at the "loop" level, while Level 2 performs vectorization on a statement-by-statement level. All vector timings for the standard benchmarks on the 3090 were done using vector Level 2. It should be noted that the SIMPLE code detected a compiler bug when run with vector Level 2. Some answers were incorrect. Subsequent decomposition of the offending loop into smaller sections produced correct answers at Level 2.

Several observations can be made from the timings.

1. Scalar 3090 execution produced results comparable to that of scalar X-MP results on programs BMK4A, BMK5, BMK14, BMK21A, BMK22, and all cases of SIMPLE. In some instances, the best 3090 results were the vector times. In a few instances, however, the best IBM numbers were obtained from scalar runs. Recall that cycle times for the two machines are, respectively, 18.5 ns for the 3090 and 9.5 ns for the X-MP. The scalar times for BMK11A and BMK11B are anomalously slow compared

with the X-MP. Neither we nor personnel at IBM can explain this anomaly at the present time.

2. In many cases, scalar and vector performance on the 3090 are nearly identical. (Exceptions to this occur on BMK8A1, BMK11A, and BMK11B.) This is in part due to the conservative nature of the Fortran compiler's economy analyzer, which generates scalar code when it cannot determine that vector code would be faster or that vector code definitely would run slower. One example is long vectors with non-unit stride. In some of these cases, it may be true that scalar mode will be faster. This is true because of the access time needed to obtain data from main memory when the data are not in cache. (See the discussion of BMK8A2 below.) Another example is for DO loops with an unknown number of trips. Since scalar mode is faster for short vectors, the economy analyzer will generate scalar code if it cannot determine that the number of trips is relatively large. However, as discussed in point 6 below, different versions of the compiler employ different algorithms for making such decisions, with large differences in performance.

3. Integer calculations, as evidenced on BMK1, are performed efficiently on the 3090.

4. As an experiment, the compiler directive "PREFER VECTOR" was used to force vectorization in one loop of HYDRO (results described elsewhere in this report), in spite of the fact that the cost analyzer predicted scalar performance would be more optimal. Indeed, the code executed more slowly in vector mode. An example of the type of loop coding that occurs often in the benchmark codes, for which the cost analyzer selects scalar operation, is given below:

```
      DO 35 J = 1,512
      TESP(J) = TEST(2,J)
   35 CONTINUE
```

Again, the non-unit stride in memory prohibits vectorization of this loop. Depending on the size of the first dimension of the variable TEST, successive fetches of 16-word blocks to access the proper values of TEST do not make vectorization cost effective.

5. Programs BMK8A1, BMK11A, and BMK11B exhibited the greatest observed speedup (2+) using vectorization.

6. Use of the older version of the VS Fortran compiler produced significant differences in programs BMK14 and BMK22 (both highly vectorizable codes). Using Version 1.1.0 in vector mode, BMK14 ran in 6.2 s and BMK22 ran in 18.2 s; these timings represent about a two-fold speedup over both the scalar mode times for both compilers and the vector mode times for the Version 2.1.0 compiler. Apparently, the newer version of the compiler was to be used with explicit vectorization directives (even though these had not been implemented at Gaithersburg) and thus was not as "aggressive" at vectorizing as was the older compiler. For example, Version 1.1.0 vectorized the SAXPY routine in BMK22, whereas Version 2.1.0 did not.

7. The rightmost column in Table I shows two entries for tuned versions of BMK14 and BMK22. Times of 1.2 and 6.3 s, respectively, represent a speedup of 6.7 and 10.0 over the scalar versions. The "tuning" consisted of replacing actual benchmark code with calls to the ESSL. This library has undergone extensive development and tuning for the

3090 architecture and is hand coded in assembly language. IBM regards it as a key item in the IBM 3090 product offering. While not tuned in the usual sense of recoding Fortran to take advantage of architectural features, the codes do indicate the performance gains that may be obtained by using ESSL. We do not expect that large portions of our scientific codes can be rewritten to take advantage of ESSL.

Overall, the vectorization speedup observed on the benchmark set was always less than three. Obtaining maximum speedup often involved recoding of loops to assure contiguous memory locations. However, the small gain in performance makes extensive code conversion an uneconomical practice in most production codes.

B. Hydro Results

Hydro is a two-dimensional Lagrangian hydrodynamics code representing codes that are a significant portion of the Laboratory workload. Two versions of the code were run on the IBM 3090: the standard version and a version optimized for contiguous vector machines in which the first dimension is the index for the inner loop. In some cases, it is not possible to interchange the order of the loops because of dependencies. IBM subsequently ran another version with a number of user subroutines integrated inline, with single-column arrays made into two-dimensional arrays the size of the mesh, and with inline Fortran code substituted for Fortran intrinsics for which the compiler does not generate vector code. This version is denoted the revised version. These modifications enabled the compiler to vectorize two loops that consume significant time; thus, we begin to see some advantage from the vector hardware. Two versions were run on the CRAY X-MP/48 (single-processor only): the standard version and an optimized version with IF blocks replaced by Cray intrinsics (necessary because of compiler vectorization inadequacies). On the IBM machine, compiler directives were applied, but the code ran slower; Cray compiler directives produced some speed gains. In Table II we report the best scalar and vector execution times for a problem size of 100 x 100 for 100 time cycles.

In our hydro runs we saw only modest performance from the vector hardware. For example, one routine ran about 10% slower in vector mode than in scalar mode. The improvement we did see came from the conversion of one function (SRCHDF, a binary search and interpolation routine) to one-dimensional arguments. As described above, substantial modification was required to obtain vectorization of two major loops, with the introduction of obscure code to accomplish the vectorization of relatively straightforward code. Speedup for those loops was about a factor of 2. Future improvements to the compiler may make these modifications unnecessary.

C. ESN Results

We also ran ESN, a deterministic particle transport code, at Gaithersburg. Although ESN is not a part of the standard benchmark set, we have been using it as a tool in parallel processing research and have collected timing data for it on a variety of machines. The code is almost entirely scalar, and the time we obtained on the 3090 is somewhat slower than that for the X-MP (see Table III). A slight performance degradation occurred on the 3090 in vector mode. (Vector Level 1 and vector Level 2 produced incorrect results with the compiler we used.) Examination of the compiler output reveals vectorization of a number of short loops (vector length 11 or less), which ran faster in scalar mode.

VI. DISCUSSION

A. Effects of Cache

We present the results from BMK8A1 and BMK8A2 in Tables IV and V with the following caveat: When comparing these results with previous benchmark reports on other machines, keep in mind that the results for the other machines represent memory-to-memory times, while the IBM 3090 times may be for cache-to-cache execution. Therefore, we regard these numbers as considerably inaccurate in describing memory-to-memory performance on the 3090. However, they do give some indication of the expected performance of codes with similar cache/memory access characteristics. To see the effect of main memory references, we refer the reader to Table V. These results are megaflop rates for various length vector operations, with a stride of 23. As an example, consider the first operation, V = V + S, where a scalar is added to each element of a vector, and the result is stored into a second vector. Note the megaflop rate falls from 11.98 for a vector length of 100 to 1.66 for a vector length of 200. How can this be understood? First, recall that the cache on the Model 200 is 64 kbytes, which in double-precision arithmetic is 8 kwords. Next, consider how the execution times (and hence the megaflop rates) are obtained for BMK8A2: an outer loop runs repeated instances of the inner loop, which does the actual vector operations. This is to give a measurably long time for the calculations. Execution time is determined by dividing the time for all the outer loops by the number of outer loops. For a stride of 23 and a vector length of 100, 2300 words of memory are required, while two such vectors span 4600 words of memory. For a vector length of 200, the two vectors span 9200 words of memory. We can now see why the execution rate drops between a vector length of 100 and 200: for the shorter vector, both vectors fit entirely within cache. Repeated executions of the inner loop do not require references to main memory once the cache is loaded the first time. For vector length of 200, however, not all of both vectors will fit in memory. Therefore, repeated execution of the outer loop would require some portion of the cache to be overwritten each time, forcing references to main memory. The performance cost is apparent.

We also note that the rates for the final vector operation, S = S + V1(I) * V2(I), are anomalously high. We believe this to be due to the compiler optimizing the code, so that it is no longer executing the proper instruction flow for valid timings (the results, of course, would still be correct).

B. Compiler Performance

Compiler technology and philosophy necessarily play a large role in determining performance on a given computer. The benchmark codes reported here were executed for comparison with results from other computers in essentially an "as-is" mode. In most instances, the only allowable changes were Fortran changes necessary to enable proper execution of the code.

Because the vector unit on the 3090 is an add-on feature to existing hardware, compatibility and price/performance ratios superseded absolute performance issues. Use of the cache for all vector operations, rather than a fast, interleaved, main memory, puts restrictions on the effectiveness of vectorization. The compiler vectorizes cautiously, basing decisions on statistics from the cost analyzer regarding the most efficient mode of operation for a given code. IBM has chosen to optimize the scalar features of its compiler for the 3090. Codes that are essentially scalar ran with times comparable to scalar execution on the CRAY X-MP. Judicious use of the compiler directives is seen by IBM as the means by which a user can override the defaults and obtain greater performance.

An example of directive use substantially improving the performance of a code can be seen in BMK14, a program consisting primarily of matrix operations. Untuned, the scalar and vector versions differ by less than 5% in their execution times because of stride problems (row access rather than column) inhibiting vectorization. By adding the compiler directive "PREFER VECTOR" to the code in the subroutine SAXPY to ignore non-contiguous memory accesses, a speedup of nearly two was obtained. As with all directives usage, specific knowledge of the particular code and parameters is necessary.

It is important to understand the characteristics of an installation's workload. Sites whose codes are primarily scalar in nature would benefit from the great effort put into scalar optimization on the 3090, with added performance in those cases where vectorization was effected. Highly vectorized workloads may involve restructuring programs in order to make use of the restrictive application of the vector units on this machine.

VII. THE VS FORTRAN MULTITASKING FACILITY (MTF)

A. Description

One objective of this benchmark trip was to gain some experience with the VS Fortran MTF. The current version of IBM's multitasking support is primitive. It allows only for task creation and synchronization via "forks" and "joins" (which are implemented in VS Fortran as "CALL DSPTCH" and "CALL SYNCH," respectively). Only the main task may fork sub-tasks. No mutual exclusion or message-passing primitives such as locks or events are available. Additionally, data sharing between the main program and any of its sub-tasks via COMMON blocks is not allowed. All common data must be explicitly passed as parameters in the DSPTCH of the sub-task routine. Although COMMON block data sharing among subprograms within a given sub-task is allowed, this fact was not made known to us until our arrival at Gaithersburg. The reason for these COMMON block implementation details is that although the 3090 hardware is itself a shared-memory architecture, multitasked jobs do not share a common memory space. Rather, an image of the part of the code to be multitasked is reproduced in memory, once for each instantiation of the sub-task. Therefore, COMMON block usage among subroutines in a sub-task appears to be approximately equivalent to the Cray Research TASK COMMON, while there is no true (global) COMMON.

In summary, we note the following inconveniences/shortcomings:

1. Programs must be re-coded to eliminate data sharing other than explicitly passed parameters, or what amounts to TASK COMMON.

2. Algorithms must be restructured to eliminate locks, events, barriers, critical sections, etc.

3. The program must be physically separated into two files, one containing the main program and all subprograms it calls, the other containing the subroutine to be "dispatched" and any subprograms it calls. Both files are compiled and loaded independently and subsequently linked together. We did not explore the question of calling a given subprogram from both the main and dispatched subroutines.

4. The VS Fortran Interactive Debug Facility, a good debugger for sequential codes, is incapable of debugging codes utilizing MTF.

B. Multitasked Codes

We originally intended to run PIC (Particle in Cell), ESN (a discrete ordinates transport code), and several examples from the IBM Multitasking Facility User's Guide. However, because of the MTF limitations described above, the recoded serial version of ESN was never successfully debugged during the trip. We did, however, use the MVS Interactive Debug Facility while debugging the new serial version, which enabled us to find several errors. Joanne Martin of IBM Yorktown has provided us with some timings from her multitasked version of ESN that she successfully ran on a Model 400. PIC and the example codes did run successfully in multitasked mode.

One problem we encountered is that the function we used for measuring CPU execution time did not work during the multitasking runs. Therefore, CPU and wall-clock* times for all MTF runs were obtained from the MVS timing data included in the dayfile for the job.

C. PIC

The version of PIC we ran using the MTF facility on the IBM 3090/200 was originally developed for use with Floating Point Systems' array processors. In this algorithm, the initialization phase can be adopted for multi-thread execution, although this was not done on the 3090. After initialization, the grid is replicated for each processor, and the particle "push," requiring greater than 95% of the total execution time, is done in parallel. At this point the tasks are synchronized and the Poisson equation is solved and electric field for the current iteration is computed.

The replicative grid scheme avoids the use of a critical section of code during the particle push and is necessary on the 3090 because of the absence of locks. However, it is not necessarily the most efficient parallel PIC algorithm.

The results for two problem sets, one processing 80000 particles and the other processing 35000 particles on a 32-by-32 grid for 60 time steps, are listed in Table VI. The multitasking speedup, shown in the third column of Table VI, is defined as the MVS CPU time divided by the MVS wall clock time. Note that the internal timing in the sequential version of PIC (which is not shown in the table) yields a result of 83.9 s on the 3090/200. The corresponding time on the CRAY X-MP/48, using the CFT 1.14 compiler, is approximately 67.0 s (obtained during production time).

The two-processor speedups for the 35000 and 80000 particle problems were 1.56 and 1.75, respectively, suggesting that multitasking on the 3090 benefits significantly from larger task granularity. We also present times from a four-processor run on a Model 400, provided by Joanne Martin.

D. IBM Examples

Results obtained multitasking several examples from the IBM Multitasking Facility User's Guide are listed in Table VII. These examples were implemented with an outer loop giving multiple trips through the inner loops. The purpose of this was to generate enough computation to take a measurable time. These examples were intended as a simple test of whether MTF worked; they do not represent codes from our workload.

*Wall clock time is defined as the total time required to execute the job.

E. ESN

Speedups for ESN are presented in Table VIII for the 3090 Model 400 and for the CRAY X-MP/48 on one, two, and four processors. These were obtained from the run times provided by Joanne Martin for the 3090. Since actual execution times were not available for the X-MP, only speedups are reported here (but see the section on the serial version of ESN above). Note that the X-MP speedups were obtained on an X-MP/48 running the Cray Operating System (COS). It appears the Cray does a somewhat better job on ESN, but we refrain from speculating on the causes as this time.

VIII. CONCLUSIONS

A. Scalar Performance

The IBM 3090 with the VS Fortran compiler delivers very good scalar performance--comparable with a CRAY X-MP in many cases. For a predominately scalar workload, we would expect the IBM 3090 to deliver excellent performance.

B. Vector Performance

The IBM 3090 Vector Facility appears to meet its design goal of delivering increased performance on some scientific and engineering codes at a small incremental cost. However, because of the 3090's cache, a large class of problems will not speed up in vector mode without significant recoding. Furthermore, the maximum speedup that can be attained without extensive hand-coding in assembly language is in the neighborhood of four times scalar, as opposed to the factors of 10 or more commonly seen on a Cray. Since many of the kernels of our large production codes are highly vectorized, we would expect them to run significantly slower on the 3090 than on the CRAY X-MP class of machines. Because absolute performance is more important to us than cost effectiveness, we cannot recommend the 3090 with Vector Facility for our large scientific codes.

C. MTF

MTF on the 3090 represents at best a first approximation to a generally usable multitasking environment. The lack of common memory, with the associated passing of shared data in parameter lists, by itself precludes implementing a production code under MTF. Furthermore, the lack of all but the most rudimentary of synchronization constructs restricts its use to a very small class of problems in which performance is not limited by load balancing, granularity, or data-sharing issues.

D. Summary

The IBM 3090 with Vector Facility is an extremely interesting machine because it combines very good scalar performance with enhanced vector and multitasking performance. For many IBM installations with a large scientific workload, the 3090/vector/MTF combination may be an ideal means of increasing throughput at minimum cost. However, as we have noted above, neither the vector nor multitasking capabilities are sufficiently developed to make the 3090 competitive with our current worker machines for our large-scale scientific codes.

IX. ACKNOWLEDGEMENTS

We thank many people at several IBM offices, including Wayne Ivester, Tejpal Chadha, and Steve Sporzynski of Gaithersburg; Doyce Nix and Deanna Skinner of Santa Fe; and Carl Ledbetter, Maria Brumbaugh, Greg Holton, Steve Thomas, Kevin Jones, Steve Hamilton, David Sol, and Troy Wilson of Kingston. Joanne Martin of IBM Yorktown also provided valuable input to this report.

Special thanks also to Tom Stup of the Los Alamos Computer Systems Group for his help in preparing the benchmark source code tapes.

This work was performed under the auspices of the U.S. Department of Energy, contract W-7405-Eng. 36.

REFERENCE

1. James H. Griffin and Margaret L. Simmons, "Los Alamos National Laboratory Computer Benchmarking 1983," Los Alamos National Laboratory report LA-10151-MS (June 1984).

Table I. Benchmark Execution Times (s)

Program Name	CRAY X-MP/48		IBM 3090		
	V	S	V	S	Tuned
1	57.9	56.9	16.8	17.4	
4A	4.3	7.9	13.5	10.7	
5	23.3	23.1	27.5	26.7	
8A1	68.6	515.7	294.8	426.8	
11A	5.7	14.9	12.4	28.5	
11B	5.4	14.5	12.8	30.6	
14	1.8	11.8	11.4	12.0	1.2
21A	7.2	7.9	7.5	7.8	
22	9.3	47.5	41.7	41.9	6.3
SIMPLE					
32X32	2.0	5.1	4.5	6.5	
64X64	8.4	21.4	18.9	26.3	
96X96	18.2	49.1	39.6	68.6	

Table II. Hydro Execution Times (s)

Machine	Standard		Optimized		Revised	
	V	S	V	S	V	S
IBM 3090	204.9	204.8	161.6	158.0	127.6	158.9
CRAY X-MP/48	52.4	103.6	20.8*	96.8*	-	-

*Slower than times reported for the X-MP benchmarks because of the library routines being used on this machine.

Table III. ESN Execution Times (s)

Machine	V	S
IBM 3090	25.4	23.7
CRAY X-MP/48	17.9	17.9

Table IV. Results of Benchmark 8A1

RATES (IN MFLOPS) ON IBM 3090 FOR *SCALAR* CODE
VECTORS ARE STORED IN CONSECUTIVE LOCATIONS
NSTEP=1

Operation	Vector Length									
	10	*25*	*50*	*100*	*200*	*500*	*1000*	*2000*	*5000*	*10000*
V=V+S	5.81	6.20	6.27	6.31	6.30	6.35	6.35	6.34	5.13	4.61
V=S*V	5.94	6.14	6.07	6.29	6.49	6.27	6.13	6.41	5.22	4.62
V=V+V	6.51	6.93	7.06	7.13	7.14	7.15	7.17	7.18	4.22	4.27
V=V*V	5.01	5.20	5.17	5.35	5.51	5.34	5.19	5.16	3.56	3.62
V=V+S*V	7.72	8.12	8.37	8.51	8.58	8.27	8.06	8.03	5.95	6.22
V=V*V+S	7.83	8.07	8.01	8.23	8.44	8.22	8.06	8.00	6.04	6.00
V=V*V+V	7.67	8.00	8.00	8.23	8.44	8.23	7.98	7.92	5.64	5.56
V=S*V+S*V	7.99	8.19	8.25	8.44	8.60	8.26	8.05	8.16	6.80	6.74
V=V*V+V*V	7.65	7.92	8.11	8.22	8.27	8.19	8.05	6.17	5.93	5.93
V=V(IND)+S	4.06	4.32	4.41	4.05	4.25	3.45	2.90	2.43	2.36	2.52
V(IND)=V*V	2.92	3.06	3.07	3.15	3.14	2.35	1.92	1.78	1.75	1.77
V(IND)=V(IND)+V*V	5.15	5.43	5.57	1.88	3.70	1.98	1.94	1.97	1.87	1.98
V=V+V*V(IND)	5.10	5.42	5.53	5.13	5.44	3.99	3.49	3.13	3.11	3.27
SUB CALLS	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.48	0.49

RATES (IN MFLOPS) ON IBM 3090 FOR *VECTOR* CODE
VECTORS ARE STORED IN CONSECUTIVE LOCATIONS
NSTEP=1

Operation	Vector Length									
	10	*25*	*50*	*100*	*200*	*500*	*1000*	*2000*	*5000*	*10000*
V=V+S	3.58	6.65	11.75	14.93	16.92	19.41	19.82	20.03	10.96	9.26
V=S*V	3.49	7.26	11.55	14.77	16.70	19.19	19.59	19.79	10.94	9.21
V=V+V	2.80	5.65	8.66	10.96	11.99	13.38	13.57	13.67	6.62	6.65
V=V*V	2.70	5.49	8.47	10.81	11.79	13.18	13.36	13.46	6.58	6.59
V=V+S*V	5.41	10.99	16.94	21.62	23.60	26.41	26.77	26.97	13.18	13.21
V=V*V+S	4.80	9.45	14.08	17.58	18.88	20.76	20.99	21.04	11.57	11.62
V=V*V+V	4.36	8.75	13.28	16.94	18.15	20.05	20.25	19.26	10.74	10.85
V=S*V+S*V	7.80	15.96	24.79	31.92	34.79	39.00	39.53	39.82	19.64	19.69
V=V*V+V*V	5.50	10.93	16.41	20.92	22.17	24.28	23.00	13.64	13.29	13.32
V=V(IND)+S	2.28	4.32	6.20	6.32	7.26	5.14	3.94	3.20	3.15	3.25
V(IND)=V*V	2.09	4.12	6.01	7.00	6.07	2.88	2.16	2.01	2.00	2.07
V(IND)=V(IND)+V*V	3.16	5.93	8.27	6.22	3.99	1.92	1.91	1.89	1.85	1.91
V=V+V*V(IND)	3.29	6.21	8.89	9.72	10.72	6.15	4.84	4.61	4.56	4.65
SUB CALLS	0.47	0.47	0.47	0.47	0.47	0.047	0.46	0.47	0.46	0.48

Table V. Results of Benchmark 8A2 for Stride at 23 (MFLOPS)

STEP=23

Operation	Vector Length						
	10	*25*	*50*	*100*	*200*	*500*	*1000*
V=V+S	3.49	7.30	10.22	11.98	1.66	0.82	0.76
V=S*V	3.30	6.96	9.90	11.75	1.65	0.82	0.76
V=V+V	2.76	5.60	7.83	6.10	0.71	0.59	0.57
V=V*V	2.69	5.49	7.72	6.05	0.71	0.59	0.57
V=V+S*V	5.30	10.85	15.32	11.93	1.42	1.19	1.14
V=V*V+S	4.76	9.41	12.99	10.73	1.40	1.18	1.13
V=V*V+V	4.47	8.94	12.54	2.47	1.05	0.88	0.88
V=S*V+S*V	7.65	15.77	22.47	17.92	2.12	1.78	1.71
V=V*V+V*V	5.59	11.10	15.61	1.96	1.15	1.09	1.09
V=V(IND)+S	2.25	4.29	3.70	2.59	0.68	0.56	0.55
V(IND)=V*V	1.89	2.99	3.72	1.22	0.50	0.44	0.44
V(IND)=V(IND)+V*V	2.98	4.71	4.99	1.15	0.71	0.70	0.69
V=V+V*V(IND)	3.29	4.63	4.98	1.21	0.76	0.71	0.71
SUB CALLS	0.49	0.49	0.49	0.49	0.49	0.49	0.51
S=S+V1(I)*V2(I)	71.38	81.36	84.96	86.49	77.42	69.70	68.58

Table VI. Results of Multitasking the PIC Code on the IBM 3090/200

A. 80000 Particle Problem

	CPU Time (s)	Wall Clock (s)	Speedup
Sequential Version (No Multitasking Calls)	86.8		1.00
4 Task Version (4 CPUs)	87.9	26.6	3.30
3 Task Version (2 CPUs)	89.7	65.1	1.38
2 Task Version "	90.4	51.6	1.75
1 Task Version "	87.2	91.2	0.96

B. 34848 Particle Problem

	CPU Time (s)	Wall Clock (s)	Speedup
Sequential Version	33.2		1.00
2 Task Version (2 CPUs)	35.4	22.7	1.56
1 Task Version "	33.4	36.8	0.91

Table VII. Results of Multitasking the MTF Examples on the IBM 3090/200

A. Example 1

	CPU Time (s)	Wall Clock (s)	Speedup
1 Task, 10000 repetitions	338.2	342.1	0.99
2 Tasks, 10000 repetitions	338.4	176.1	1.92
1 Task, 1000 repetitions	34.0	36.5	0.93
2 Tasks, 1000 repetitions	34.0	20.33	1.67
1 Task, 100 repetitions	3.6	6.3	0.57
2 Tasks, 100 repetitions	3.6	4.9	0.73

B. Example 2

	CPU Time (s)	Wall Clock (s)	Speedup
1 Task, 10000 repetitions	271.0	274.6	0.99
2 Tasks, 10000 repetitions	271.0	143.5	1.89
1 Task, 1000 repetitions	27.2	30.13	0.90
2 Tasks, 1000 repetitions	27.2	17.3	1.57
1 Task, 100 repetitions	2.9	5.3	0.55
2 Tasks, 100 repetitions	2.8	4.4	0.64

Table VIII. ESN Multitasking Speedups: IBM 3090/400 and CRAY X-MP/48

	3090 Speedup	X-MP Speedup
1 Task	0.97	1.00
2 Tasks	1.88	1.99
4 Tasks	3.17	3.71